PIOMan: a Generic Framework for Asynchronous Progression and Multithreaded Communications

Alexandre DENIS – Alexandre.Denis@inria.fr

Inria Bordeaux – Sud-Ouest, France

PIOMan *is a generic framework to be used by MPI implementations. It brings seamless asynchronous progression of communication by opportunistically using available cores. It uses system threads and is composable with any multithreaded runtime system.*

Multithreaded Communication Engine

Asynchronous progression

- Parallelize communications
- Decompose communication library in basic tasks aka **tasklets**
- Schedule tasklets on available cores

Schedule tasklets from user space on:

- **Idle** - for opportunistic resource usage
- **Timer** – for guaranteed progression
- **Explicit polling** – for progression at least as efficient as without tasklets

Contention-free task scheduling

High frequency task scheduling causes

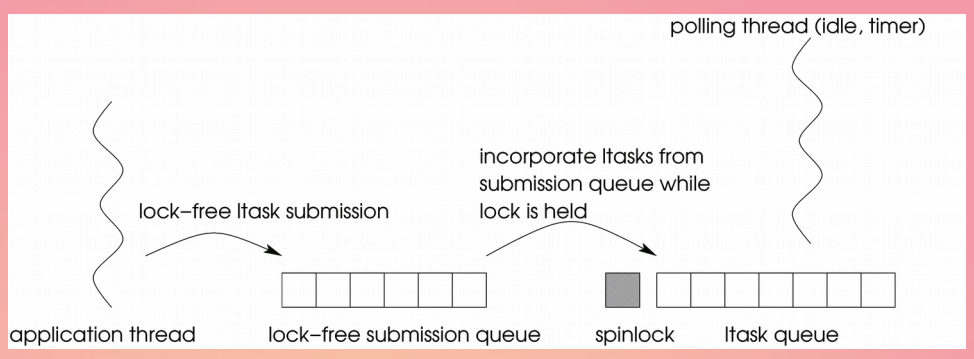
- **Contention** on data structures
- Competition between polling and task submission

Locality-aware scheduling

- Hierarchical tree of tasks lists based on hwloc
- Submit tasks in local list
- High frequency polling on tree leaves, reduce frequency on parents

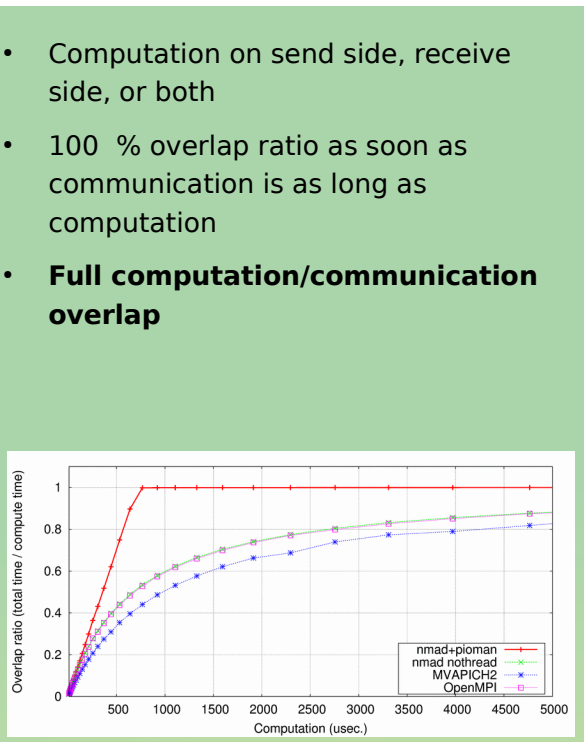
Advanced locking scheme

- **Mutex**- competition to acquire lock
- **Lock-free**- competition to commit (compare-and-swap), competition on cacheline
- **Submission queue**- our proposal: separate lists for tasks submission and polling
- Lock-free submission, spinlock for polling
- Mitigates contention between polling threads, and between polling and submission

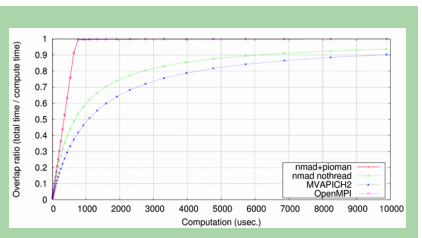


Progression Benchmarks

- Computation on send side, receive side, or both
- 100 % overlap ratio as soon as communication is as long as computation
- **Full computation/communication overlap**

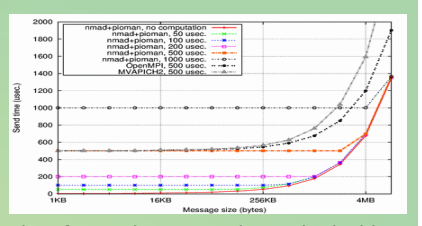


Communication/computation overlap ratio, computation on both sides



Overlap ratio, computation on sender side

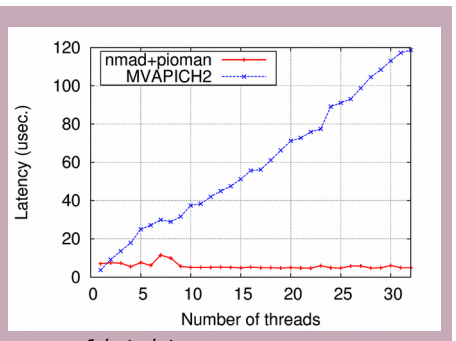
Overlap ratio, computation on receiver side



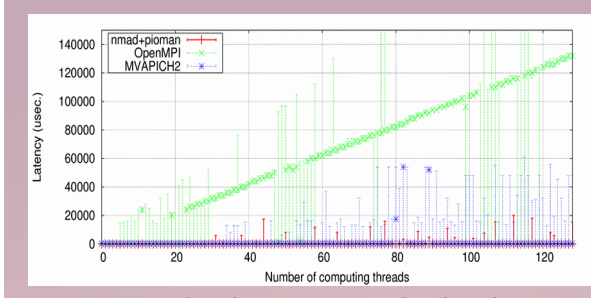
Time for send, computation on both sides

Multithreaded Benchmarks

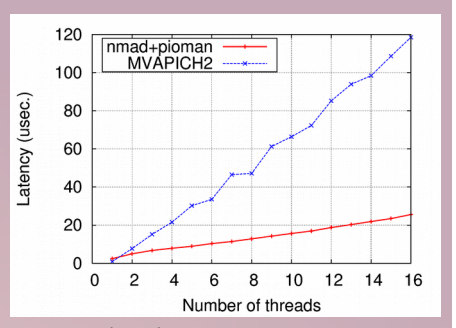
- Communication progresses consistently even with other threads
- **Constant-time reactivity**
- Scalable with number of threads
- Better multi-threading behavior than state-of-the-art MPI implementations



1 byte latency
1 thread to N threads



1-to-1, 1 byte latency, N computing threads

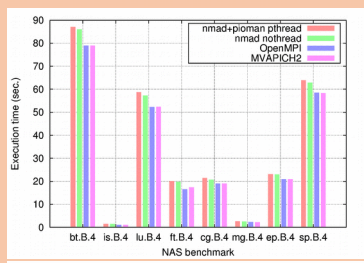


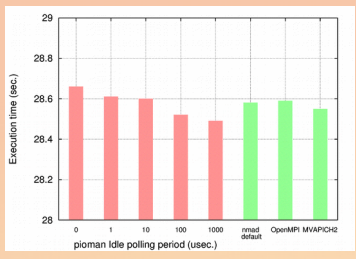
1 byte latency
N threads to N threads

Overhead

- **Low overhead** due to multithreading and polling

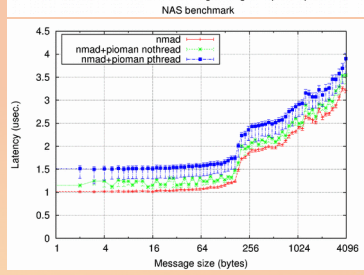
Comparison of NAS benchmarks





NAS EP benchmark for varying polling frequency

Latency overhead



http://runtime.bordeaux.inria.fr/pioman/